

Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas

Organización del Computador 2

Segundo parcial – 19/11/2015

A+

1 (30)	2 (45)	3 (25)	
30	42	25	97

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Ej. 1. (30 puntos)

Se tiene la siguiente tabla GDT:

Indice	Base	Límite	DB	S	P	L	G	DPL	Tipo
1	0x00000000	0x00800	1	1	1	0	0	0	0x8
2	0xF0000000	0x0FFFF	1	1	1	0	1	2	0xB
3	0x00120000	0x00000	1	1	1	0	1	3	0x2
4	0x001FFFFF	0x00100	1	1	1	0	1	2	0x0

Y el siguiente esquema de paginación:

Rango Lineal	Rango Físico	Atributos
0x00000000 - 0x00000FFF	0x00010000 - 0x00010FFF	read only, supervisor
0xF0F55000 - 0xF0F57FFF	0x00022000 - 0x00024FFF	read/write, supervisor
0x00123000 - 0x00124FFF	0x00005000 - 0x00006FFF	read/write, user

(12p) a. Especificar todas las entradas de las estructuras necesarias para construir un esquema de paginación. Suponer que todas las entradas no mencionadas son nulas. 12

(18p) b. Resolver las siguientes direcciones, de lógica a lineal y a física. Utilizar las estructuras definidas y suponer que cualquier otra estructura no lo está. Si se produjera un error de protección, indicar cuál error y en qué unidad. Definir EPL en los accesos a datos. El tamaño de todas las operaciones es de 2 bytes.

- I - 0x08:0x000008FF - CPL 00 - ejecución
- II - 0x13:0x00020000 - CPL 01 - lectura 18
- III - 0x1B:0x00005000 - CPL 11 - escritura
- IV - 0x10:0x00F55555 - CPL 10 - lectura
- V - 0x20:0x00004200 - CPL 00 - escritura
- VI - 0x21:0x000EA4E0 - CPL 01 - lectura

recordar:

4KB=0x1000, 64KB=0x10000, 1MB=0x100000, 4MB=0x400000,
1GB=0x40000000, 2GB=0x80000000, 3GB=0xC0000000, 4GB=1=0xFFFFFFFF.

Ej. 2. (45 puntos)

Se desea implementar un sistema multi-tareas semi-colaborativo. El mismo cuenta con un kernel que hace correr concurrentemente a 8 tareas de usuario independientes. Las mismas no deben poder leer ni escribir la memoria de las demás. Las mismas corren hasta que pasen K ciclos o decidan ceder el control. Además, cuando el usuario presione la tecla A se le otorgarán K ciclos extra a la tarea actual (solamente por el período de ejecución actual). Cuando se agota el tiempo, el scheduler pasa a la siguiente tarea. Para ceder el control voluntariamente, en cambio, las tareas utilizarán el syscall 55, indicando en EAX cuál quieren que sea la próxima tarea. Asumir que nunca se generan excepciones y que una tarea no puede cederse el control a sí misma.

- (15p) a. Detallar los campos relevantes de todas las estructuras involucradas en el sistema para manejar segmentación, tareas, interrupciones y privilegios. Instanciar las estructuras con datos y explicar su funcionamiento. Describir el esquema de segmentación que se utilizará así como también el de paginación si es que lo utiliza. **15**
- (20p) b. Escribir el código de las rutinas de atención de interrupciones del reloj y el syscall 55. **17**
- (10p) c. Escribir el código de la rutina de atención de teclado. **10**

Nota: Cualquier código pedido debe estar escrito en C o ASM. Asumir que se dispone de las definiciones de las estructuras del procesador en C. El scancode de la tecla A es 0x1E y el puerto del teclado es el 0x60.

Ej. 3. (25 puntos)

Se tiene un sistema ya funcionando con segmentación *flat* y paginación activada donde corren 8 tareas. Se desea que éstas puedan comunicarse con memoria compartida. Para tal fin, se agregará al sistema el syscall 60, que devuelve a la tarea usuario la dirección de una página para lectura o escritura. Esta syscall recibirá en EAX si es lectura o es escritura y en EBX el ID de la tarea destino, y devolverá en EAX la dirección de la página otorgada. Para implementar el manejo se escribirá una rutina ASM básica que maneje la interrupción y llame a otra de más alto nivel escrita en C que realice lo deseado.

- (4p) a. Describir que hay que agregar o cambiar al sistema para implementar el syscall. **4**
- (6p) b. Escribir el código ASM de manejo del syscall 60. **6**
- (15p) c. Escribir el código de la rutina `void* mapear(uint es_lectura, uint id_destino)`. **15**

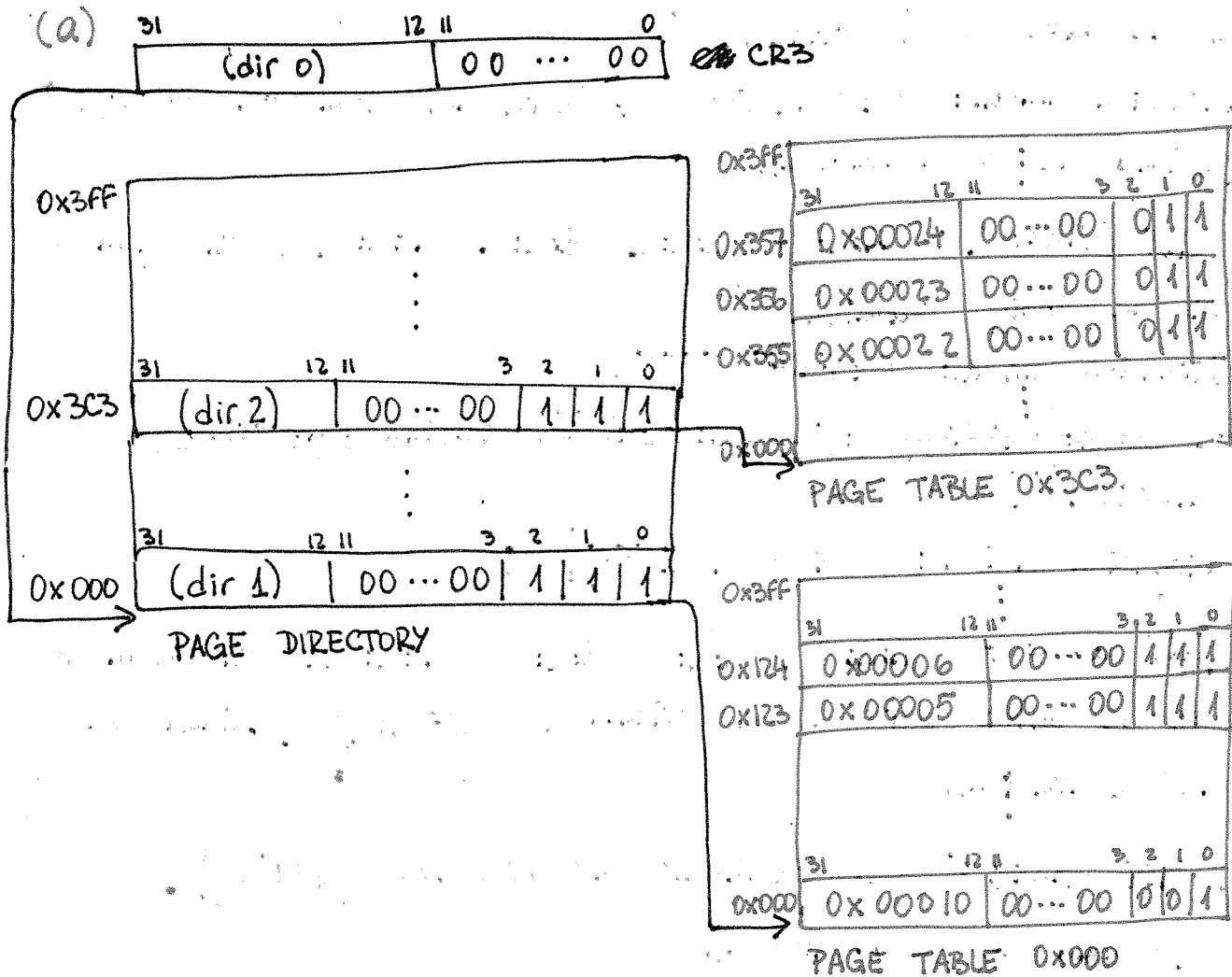
Para resolver el ejercicio, se cuenta con algunas rutinas:

- `int tarea_actual()` que devuelve el ID de la tarea actual.
- `int mapear_pagina(uint virtual, uint fisica, uint cr3, uint atributos)`.
- `void* dame_fisica_libre()` que devuelve y asigna una dirección de una página libre en memoria física.
- `void* dame_virtual_libre(uint tarea)` que devuelve y asigna una dirección de una página libre en la memoria virtual de la tarea especificada.

Puede asumir que cuando una tarea pide para lectura de otro proceso, este otro ya pidió para escritura antes. Además, sólo se puede compartir una página por cada par de tareas.

① Segmentos:

	BASE	LIMIT	G	LIMIT (BYTES)	END	DPL	TYPE
1	0x0000 0000	0x00800	0	0x0000 0800	0x0000 0800	0	code, non-read.
2	0xF000 0000	0x0FFFF	1	0x0FFF FFFF	0xFFFF FFFF	2	code, readable
3	0x0012 0000	0x0000	1	0x ⁰⁰⁰⁰ 0FFF DFFF	0x0012 0FFF	3	data, writable
4	0x001F FFFF	0x00100	1	0x0010 0FFF	0x0030 0FFE	2	data, non-writ.



Harán falta las dos entradas esquematizadas en el Page Directory, y las tres que aparecen en cada Page Table. ~~(page 0)~~ (dir 0), (dir 1) y (dir 2) hacen referencia a los 20 bits más significativos de las direcciones base del Page Directory y las dos Page Tables, respectivamente.

(b) I. GDT index: 1. RPL: 0 CPL: 0 DPL: 0. El segmento es de código ✓ *antes de calcular la dirección lineal calcula el límite*

Dirección lineal: 0x0000 08FF → fuera del segmento. #GP ✓
(unidad de segmentación).

I. GDT index: 2. RPL: 3 CPL: 1 EPL: 3 DPL: 2 × #GP ✓
(unidad de segmentación).

III. GDT index: 3 RPL: 3 CPL: 3 EPL: 3 DPL: 3. El segmento es de datos y es writable ✓.

Dirección lineal: 0x0012 5000 → fuera del segmento. #GP ✓
(unidad de segmentación).

IV. GDT index: 2 RPL: 0 CPL: 2 EPL: 2 DPL: 2. El segmento es de código readable. ✓

Dirección lineal: 0xF0F5 5555.

PD index: 0x3C3, PT index: 0x355. Página read/write, Supervisor ✓.

Dirección física: 0x0002 2555 ✓

V. GDT index: 4 RPL: 0 CPL: 0 EPL: 0 DPL: 2. El segmento es de datos pero no es writable. #GP (unidad de segmentación).

VI. GDT index: 5 → segmento no presente #NP ✓
(unidad de segmentación).

② (a) Para manejar segmentación hará falta una GDT, que será un arreglo de entradas de GDT (descriptores). Dado que se usará un esquema de segmentación flat, harán falta 4 descriptores en la GDT que referencien a segmentos que abarquen toda la memoria a usar, de código y de datos, con DPL 0 y 3.

También en la GDT deberá haber entradas ~~para~~ los descriptores de TSS (task state ~~segments~~) que permitirán aislar los tareas. Deberemos disponer de 9 descriptores de TSS en la GDT: uno para la tarea inicial, que permitirá iniciar la primera tarea, y 8 para las tareas propiamente dichas. Harán falta 9 TSS ~~estructuras~~, además, es decir estructuras donde almacenar el contexto del procesador al cambio de tarea.

Para manejar interrupciones necesitaremos una IDT, que también será un arreglo de descriptores. Allí definiremos un Interrupt Gate Descriptor para cada interrupción que queramos atender: teclado, reloj, syscall 55.

Como queremos evitar que las tareas puedan ~~leer~~ leer o escribir la memoria de los demás, usaremos distintos esquemas de paginación para cada uno. Para esto necesitaremos un Page Directory*, tantas Page Tables como hagamos falta para mapear el kernel con identity mapping (así su código puede ejecutarse desde

* uno para el kernel y uno para cada una de las 8 tareas.

cualquier tarea) y un contador de páginas libres que nos permita ~~crear~~ crear nuevas tablas de páginas y asignar más memoria a las tareas que lo requieran. ✓

Por último hará falta una serie de estructuras para el funcionamiento del scheduler: la id de la tarea que se encuentra en ejecución, una lista de todas las tareas ~~en ejecución~~ con su id y el ~~id~~ selector de su TSS en la GDT, un contador de ciclos para saber cuando se acaba el tiempo de una tarea.

Como esquema de paginación, podemos mapear la

[illegible]

cada tarea el código del kernel con identity mapping y privilegio de supervisor, y a continuación las páginas que le fueron asignados para código, datos y pila, cuidando de que al inicializar la tarea estas direcciones se reflejen en la TSS. ✓

(b) • RUTINA PRINCIPAL (en ASM) DEL RELOJ.

```
pushad  
call pic-fin-int  
- Cmp [ciclos-restantes], 0x0  
jne .fin-1
```

- aviso al PIC que atendi la interr.
- a la tarea le quedan ciclos?

jne .fin-1

Call sched - tarea - siguiente

pregunto por la tarea sig.

W. J. R. R.

str. ex

cmp ax, cx

je .fin-2

me fijo de no saltar a la misma tarea.

~~jmp~~

mov [tss-selector], ax

jmp far [tss-~~selector~~]
offset

; salto a la tarea siguiente

jmp fin2 → 0/0: no me enter

.fin_1:

dec [ciclos-restantes] ^{reinciendo los ciclos-restantes}

; si quedaban ciclos

; ahora queda un ciclo menos

popad

iret

Para lo
torio
siguiente

; pusho registros

; retorno

; vengo de otra tarea

; reinicio los k ciclos

.fin_2:

mov [ciclos-restantes], k

; ~~pusho~~ recupero registros

popad

iret

; retorno

• EN LA SECCIÓN DE DATOS:

tss-offset: dd 0x0000 0000

tss-selector: ~~dd~~ dw 0x0000

• RUTINA PRINCIPAL DE LA SYSCALL 55 (en ASM):

pushad

; pusho registros

; aviso al PIC ~~no~~ (mentira, no le

; aviso modo)

~~cmp~~

push eax

call sched-~~fin~~ ^{tss-selector}; pregunto el selector de tss,
de la tarea que me pasaronadd ~~esp~~ esp, 4

str cx

cmp ax, cx

je .fin

; me fijo de no saltar a la
; misma tarea

mov [tss-selector], ax

jmp far [tss-offset]

; salto a la tarea pedida

mov [ciclos_restantes], k

¡ acá vengo de otra tarea
¡ reinicio los ciclos restantes

fin:

~~popad~~
iret

¡ recupero los registros
¡ retorno

● FUNCIONES AUXILIARES DEL SCHEDULER (en C).

// por compatibilidad se omiten tildes.

unsigned short sched_tarea_siguiente() { // devuelve el selector
// de TSS de la proxima
// tarea y actualiza el
// scheduler

~~if (scheduler.tarea_actual == 7) {~~
~~return~~

if (scheduler.tarea_actual == 7) {

~~return scheduler.tareas[0].tss_selector;~~
~~} else {~~

~~return scheduler.tareas[scheduler.tarea_actual~~

scheduler.tarea_actual = 0; // si lleve al final del arreglo
// vuelvo al principio

} else {

scheduler.tarea_actual++; // actualizo el scheduler

}

return scheduler.tareas[scheduler.tarea_actual].tss_selector;

}

unsigned short sched_tss_selector (unsigned int task_id) {

~~int i;~~ // dado un id de tarea devuelve su selector
~~int i;~~ // de TSS y actualiza el scheduler

int i;

for (i=0; i<8; i++) {

if (scheduler.tareas[i].id == task_id) {
scheduler.tarea_actual = i; // actualizo scheduler

return scheduler.tareas[i].tss_selector; // devuelvo
// selector de TSS

}

}


```

return 0 0; // si la tarea no existia devuelvo 0
// (se rompe todo)
}

```

• ESTRUCTURA DEL SCHEDULER:

```

typedef struct sched_t { // estructura de scheduler

```

```

    unsigned char tarea_actual;
    sched_tarea tareas[8];

```

```

} sched_t;

```

```

typedef struct sched_tarea { // entradas en la lista de tareas.

```

```

    unsigned short tss_selector;
    unsigned int id;

```

```

} sched_tarea;

```

• VARIABLES GLOBALES:

```

extern unsigned int int ciclos_restantes;
extern sched_t scheduler;

```

```

(c) pushad                ; pusho registros
    call pic_fin_int       ; aviso al PIC
    in ax, 0x60            ; lee el scan code
    cmp ax, 0x1E          ; me fijo si fue la A
    jne .fin              ; si no, retorno
    add diros [ciclos_restantes], K ; sumo K ciclos
    .fin:
        popad              ; recupero registros
        iret               ; retorno

```

Nota: Asumo que esté definida la rutina pic_fin_int, que comunique al PIC el fin de la interrupción.

③ (a) Hay que agregar, en el índice 60 de la IDT, un Interrupt Gate ~~que apunte a la rutina~~ Descriptor que apunte a la rutina que escribiremos. Es importante que su DPL sea 3 o los tareas no podrán realizar la syscall. (ver nota ~~de~~ del inciso (c)).

(b) `pushad` ; pushar registros
`push ebx` ; id_destino
`push eax` ; es_lectura
`call mapear` ; eax = dirección a retornar
`mov [temp], eax` ; muevo eax a una variable temporal
`popad` ; recupero registros
`mov eax, [temp]` ; recupero eax
`iret` ; retorno

SECCIÓN DE DATOS:

~~temp: dd 0~~

`temp: dd 0` ; variable temporal

(c) `void* mapear (uint es_lectura, uint id_destino) {`

~~`void* fisica = dame-fisica-libre();`~~

~~`uint cr3 = cr3();`~~

`void* virtual = dame-virtual-libre (tarea-actual());`

`if (es_lectura) {`

`void* fisica = paginas-compartidos [id_destino][tarea-actual];`

~~`uint cr3 = cr3();`~~

`mapear-pagina (virtual, fisica, cr3, PAGE-ATTR-USER | PAGE-ATTR-PRESENT);`

`} else {`

`void* fisica = dame-fisica-libre();`

`mapear-pagina (virtual, fisica, cr3, PAGE-ATTR-USER |`

`PAGE-ATTR-READ-WRITE | PAGE-ATTR-PRESENT);`

`paginas-compartidos [tarea-actual()][id_destino] = fisica;`

`}`

`return virtual;`

Nota 2: Asumo la existencia de una rutina,
uint rcr3(void) que lee el valor contenido en CR3.

```
# define PAGE_ATTR_USER
# define PAGE_ATTR_READ_WRITE
# define PAGE_ATTR_PRESENT
```

0x3

0. x 1

no dye node